

Manejo de Excepciones en Java y Genericidad

Estructura de Datos/Algoritmos I

Depto. de computación-UNRC

Prof. Pablo Castro

Excepciones y Genericidad

Muchas veces los procedimientos (o métodos) están definidos solo para alguna parte de sus dominios. Por ejemplo:

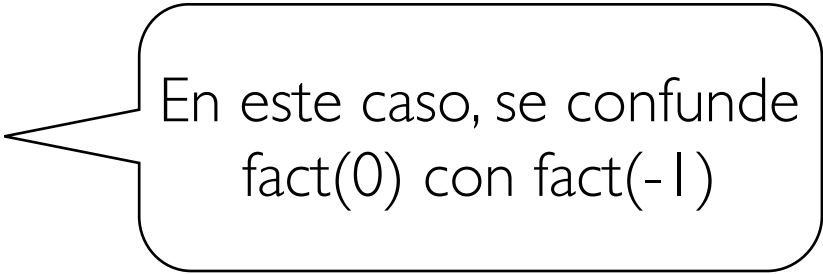
```
/**
 * A method to calculate the factorial
 * @param n the number for the factorial
 * @pre n >= 0
 * @post the factorial
 */
public static int fact(int i){
    if (n == 0)
        return 1
    else
        return (n * fact(n-1));
}
```

Cuando la precondición es falsa, el método puede tener cualquier funcionamiento...

Procedimientos parciales

Por ejemplo, podríamos retornar un valor especial:

```
/**
 * A method to calculate the factorial
 * @param n the number for the factorial
 * @pre n >= 0
 * @post the factorial
 */
public static int fact(int i){
    if (n <= 0)
        return 1
    else
        return (n * fact(n-1));
}
```



En este caso, se confunde
fact(0) con fact(-1)

Notar que:

- La llamada a factorial con números negativos es un error.
- Retornar valores especiales en el caso de errores puede hacer que se confundan con aquellos validos.

Tratamiento de errores

Lo que buscamos es un mecanismo que nos permita distinguir las ejecuciones anormales de las normales. Las excepciones nos permiten:

- Terminar un procedimiento normalmente, o
- Terminar un procedimiento de forma anormal.

Hay varios tipos de excepciones, en Java cada tipo de terminación anormal pertenece a algún tipo de excepción predeterminada.

Excepciones en java

Un procedimiento que puede terminar excepcionalmente es especificado con la palabra *throws*:

```
public static int fact(int n) throws NonPositiveException{
```

Significado:

- El método *fact* puede terminar con una excepción.
- En este caso la excepción es de tipo: *NonPositiveException*.

Se pueden tirar varios tipos de excepciones:

```
/**
 * A method to search in a given array
 * @param    a    the array
 * @param    i    the element to search
 * @return    the position of the element
 * @pre 0 <= i < a.length
 * @post the array is not modified
 * in case of not found it throws an exception
 */
public static int search(int[] a, int i) throws NotFoundException{
    // TBD
}
```

Excepciones en JAVa

Observaciones:

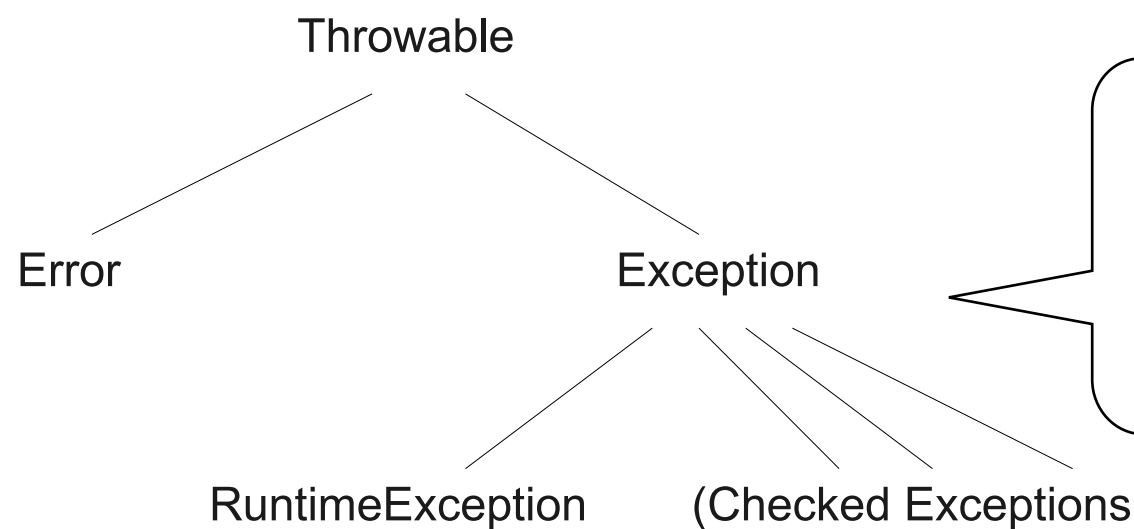
- El perfil del procedimiento debe dejar en claro cuales excepciones puede tirar el procedimiento.
- En la documentación se debe indicar la causa de cada excepción.
- Cuando un método es local a una clase (no se usa afuera) no hacen falta las excepciones.

Excepciones en JAVA

Las excepciones en Java son subtipos de:

- La clase *Exception*, o
- La clase *RuntimeException*.

Es una clase de excepción que indica que algo grave sucedió y el programa no se puede recuperar



Cuando definimos excepciones estas deben ser subclases de *Runtime* o *CheckedException*

Tipos de Excepciones

- Las subclases de *RuntimeException* son llamadas **no chequeables**.
- Las que no son subclases de *RuntimeException* son llamadas **chequeables**.

Las diferencias son:

Las excepciones no chequeables se pueden poner o no en el perfil, las ponemos solo cuando es necesario

- Cuando un procedimiento puede tirar una excepción chequeable, debe ser especificado en el perfil del método.
- Las excepciones chequeables deben ser manejadas o capturadas (sentencia *catch*).

Excepciones en JAVA

Cuando definimos una excepción debemos definir que tipo es:

```
public class MiExcepcion extends Exception{  
    public MiExcepcion(){  
        super();  
    }  
    public MiExcepcion(String s){  
        super(s);  
    }  
}
```

Cuando queremos una excepción no chequeable definimos:

```
public class MiExcepcion extends RuntimeException{  
    public MiExcepcion(){  
        super();  
    }  
    public MiExcepcion(String s){  
        super(s);  
    }  
}
```

Uso de excepciones

Cuando creamos una excepción podemos indicar cual es el problema.

```
MiExcepcion e = new MiExcepcion();  
// o bien  
MiExcepcion e = new MiExcepcion("Mensaje");
```

- Cuando definimos una excepción la ponemos en el mismo paquete que las clases que la usan.
- Cuando creamos una excepción podemos poner un mensaje explicando lo que sucedió.
- Los nombres de las excepciones deben ser explicativos.

Uso de Excepciones

Un ejemplo:

```
/**
 * A method to calculate the factorial
 * @param n the number for the factorial
 * @pre n >= 0
 * @post the factorial
 * in the case of n<0 its throws an exception
 */
public static int fact(int n) throws NonPositiveException{
    if (n < 0)
        throw new NonPositiveException("Fact: Negative Number");
    else{
        if (n == 0)
            return 1;
        else
            return (n * fact(n-1));
    }
}
```

Manejando Excepciones

Cuando un procedimiento dispara una excepción el hilo de ejecución es transferido a algún lugar en donde se maneja esa excepción.

```
try{  
    x = fact(y);  
}  
catch(Exception e){  
    // aca podemos usar e  
}
```

En la parte *catch* escribimos lo que debe ejecutarse cuando ocurre una excepción.

```
try{  
    y = Arrays.search(x);  
}  
catch(Exception e){  
    System.out.println(e);  
    return;  
}
```

El manejo de errores lo hace el **cliente** y no la librería!

Cuando usar excepciones

Debemos usar excepciones para:

- Tratar los casos en que la precondition no es `true`. Cuando esto no es muy costoso!
- Evitar codificar ejecuciones anormales como normales.
- Las excepciones permiten desarrollar software más robusto: *incluso bajo la ocurrencia de errores el sistema sigue comportandose de una forma aceptable.*

Chequeables vs no-chequeables

- No chequeables cuando:
 - Proviene de errores de programación (variables *null*)
 - Hay formas simples de evitar las excepciones.
- Chequeables cuando:
 - Tenemos situaciones recuperables y no provienen de errores de programación (entrada/salida).

Chequeables vs no-chequeables

- Las excepciones no-chequeables no necesitan estar dentro de un *try*.
- Los métodos con excepciones chequeables deben estar dentro de *try()*, y además deben listar estas excepciones en la cláusula *throws*.

Solo listamos las excepciones no-chequeables cuando consideremos importantes documentarlas.

Genericidad

Hasta ahora utilizamos la clase *Object* para obtener polimorfismo, sin embargo:

- En general los TADS son homogéneos: siempre utilizan el mismo tipo de datos.
- Si usamos *Object*, tenemos que utilizar casting lo cual puede producir errores en tiempo de ejecución.

En Java podemos evitar estos problemas usando genericidad.

Un Ejemplo

```
/**
 * Una Lista Generica Simple
 * @author Pablo Castro
 */
public interface Lista<E>{

    /**
     * Dice si la lista es vacia o no
     */
    public boolean esVacia();

    /**
     * Devuelve la longitud de la lista
     */
    public int longitud();

    /**
     * Vacis al lista
     */
    public void vaciar();

    /**
     * Agrega un item en la posicion i
     */
    public void insertar(int i, E item) throws ExcepcionLista, IndiceFueraDeRangoLista;

    /**
     * Elimina el i-esimo elemento de la lista
     */
    public void eliminar(int i) throws IndiceFueraDeRangoLista;

    /**
     * Dice si el elemento esta en la lista o no
     */
    public E obtener(int i) throws IndiceFueraDeRangoLista;
}
```

Clase genérica, el tipo E debe ser instanciado

Un Ejemplo de Uso

Para utilizarla podríamos hacer:

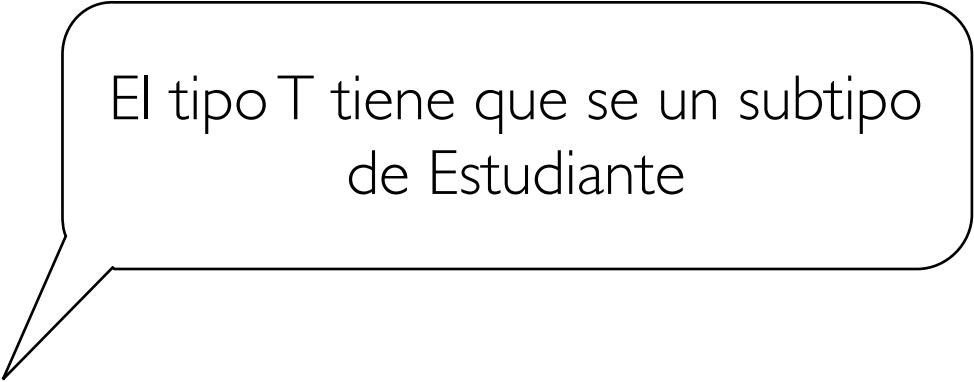
```
// Un Ejemplo simple de uso
// de las listas genericas
public class TestLista{
    static public void main(String[] args){
        ListaArray<String> l = new ListaArray<String>();
        l.insertar(1,"primer");
        l.insertar(2,"segundo");
        String elem = l.obtener(2);
        System.out.println(elem);
    }
} // final de ejemplo
```

No hace falta castings!

Todos los elementos de la lista van a ser Strings, se logra una genericidad de una forma homogénea

Genericidad y Herencia

Muchas veces necesitamos usar clases que tengan una superclase en particular:



El tipo T tiene que ser un subtipo de Estudiante

```
// Un ejemplo de clase que utiliza genericidad restringida
public interface Registration<T extends Estudiante>{

    public void registrar(T estudiante, Curso c);
    public void dejar(T estudiante, Curso, c);

}
```

En este caso el tipo T debe ser un subtipo del tipo, o clase, Estudiante.

Ejercicio: **Implementar el tipo Lista con genericidad.**